Parallel Processing: Multiprocessor Systems (MIMD)

- MP - A computer system with at least 2 processors:
  - Processor
  - Cache
  - Interconnection Network
  - Memory
  - I/O

  - Q1 – How do they share data?
  - Q2 – How do they coordinate?
  - Q3 – How many processors can be supported?

Shared Memory Multiprocessor (SMP)

- Q1 – Single address space shared by all processors/cores
- Q2 – Processors coordinate/communicate through shared variables in memory (via loads and stores)
  - Use of shared data must be coordinated via synchronization primitives (locks) that allow access to data to only one processor at a time
- All multicore computers today are SMP

CS10 Review: Higher Order Functions with “CS10: Sum Reduction”

- Useful HOFs (you can build your own!)
  - map Reporter over List
  - keep items such that Predicate from List

- combine with Reporter over List

CS61C Example: Sum Reduction

- Sum 100,000 numbers on 100 processor SMP
  - Each processor has ID: 0 ≤ Pn ≤ 99
  - Partition 1000 numbers per processor
  - Initial summation on each processor [Phase I]
  - Aka, “the map phase”
    - sum[Pn] = 0;
    - for ( self = 1000*Pn; self < 1000*(Pn+1); self = self + 1)

  - Now need to add these partial sums [Phase II]
    - Reduction: divide and conquer in “the reduce phase”
      - Half the processors add pairs, then quarter, ...
      - Need to synchronize between reduction steps
Example: Sum Reduction

Second Phase:
After each processor has computed its “local” sum
This code runs simultaneously on each core

half = 100;
repeat
  synch();
  /* Proc 0 sums extra element if there is one */
  if (half/2 := 0 \&\& Pn == 0)
    sum[0] = sum[0] + sum[half-1];
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
  until (half == 1);

An Example with 10 Processors


half = 10

half = 5

half = 2

half = 1

Peer Instruction

half = 100;
repeat
  synch();
  /* Proc 0 sums extra element if there is one */
  if (half/2 := 0 \&\& Pn == 0)
    sum[0] = sum[0] + sum[half-1];
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
  until (half == 1);

What goes in Shared? What goes in Private?

<table>
<thead>
<tr>
<th>half</th>
<th>sum</th>
<th>Pn</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>PRIVATE</td>
<td>PRIVATE</td>
</tr>
<tr>
<td>(b)</td>
<td>PRIVATE</td>
<td>PRIVATE</td>
</tr>
<tr>
<td>(c)</td>
<td>PRIVATE</td>
<td>PRIVATE</td>
</tr>
<tr>
<td>(d)</td>
<td>SHARED</td>
<td>PRIVATE</td>
</tr>
<tr>
<td>(e)</td>
<td>SHARED</td>
<td>SHARED</td>
</tr>
</tbody>
</table>
Three Key Questions about Multiprocessors

- Q3 – How many processors can be supported?
- Key bottleneck in an SMP is the memory system
- Caches can effectively increase memory bandwidth/open the bottleneck
- But what happens to the memory being actively shared among the processors through the caches?

Shared Memory and Caches

- What if?
  - Processors 1 and 2 read Memory[1000] (value 20)
  - Processor 0 writes Memory[1000] with 40

Keeping Multiple Caches Coherent

- Architect’s job: shared memory ➞ keep cache values coherent
- Idea: When any processor has cache miss or writes, notify other processors via interconnection network
  - If only reading, many processors can have copies
  - If a processor writes, invalidate all other copies
- Shared written result can “ping-pong” between caches

How Does HW Keep $ Coherent?

Each cache tracks state of each block in cache:

- Shared: up-to-date data, not allowed to write
- Modified: up-to-date, changed (dirty), OK to write
  - no other cache has a copy, copy in memory is out-of-date
- Invalid: Not really in the cache

2 Optional Performance Optimizations of Cache Coherency via new States

Exclusive: up-to-date data, OK to write (change to modified)
  - no other cache has a copy, copy in memory up-to-date
  - Avoids writing to memory if block replaced
  - Supplies data on read instead of going to memory

Owner: up-to-date data, OK to write (if invalidate shared copies first then change to modified)
  - other caches may have a copy (they must be in Shared state)
  - copy in memory not up-to-date
  - So, owner must supply data on read instead of going to memory

http://youtu.be/WdRqzqPfdM
Common Cache Coherency Protocol: MOESI (snoopy protocol)

• Each block in each cache is in one of the following states:
  - Modified (in cache)
  - Owned (in cache)
  - Exclusive (in cache)
  - Shared (in cache)
  - Invalid (not in cache)

Compatibility Matrix: Allowed states for a given cache block in any pair of caches

Common Cache Coherency Protocol: MOESI (snoopy protocol)

• Each block in each cache is in one of the following states:
  - Modified (in cache)
  - Owned (in cache)
  - Exclusive (in cache)
  - Shared (in cache)
  - Invalid (not in cache)

Compatibility Matrix: Allowed states for a given cache block in any pair of caches

Cache Coherency and Block Size

• Suppose block size is 32 bytes
• Suppose Processor 0 reading and writing variable X, Processor 1 reading and writing variable Y
• Suppose in X location 4000, Y in 4012
• What will happen?
  - Effect called *false sharing*
  - How can you prevent it?

Dan’s Laptop? sysctl hw

be careful! you can *change* some of these values with the wrong flags!

And In Conclusion, ...

• Sequential software is slow software
  - SIMD and MIMD only path to higher performance
• Multiprocessor (Multicore) uses Shared Memory (single address space)
• Cache coherency implements shared memory even with multiple copies in multiple caches
  - False sharing a concern
• Next Time: OpenMP as simple parallel extension to C